Moment Based Painterly Rendering Using Connected Color Components

---

A thesis

submitted in partial fulfilment

of the requirements for the degree

of

Master of Science

in

Computer Science and Software Engineering

in the

University of Canterbury

by

Mohammad Obaid

Examining Committee

Dr R. Mukundan	Supervisor
Associate Professor Tim Bell	Co-Supervisor

University of Canterbury

2006

To my Mother and Father

Abstract

Research and development of Non-Photorealistic Rendering algorithms has recently moved towards the use of computer vision algorithms to extract image features. The feature representation capabilities of image moments could be used effectively for the selection of brush-stroke characteristics for painterly-rendering applications. This technique is based on the estimation of local geometric features from the intensity distribution in small windowed images to obtain the brush size, color and direction. This thesis proposes an improvement of this method, by additionally extracting the connected components so that the adjacent regions of similar color are grouped for generating large and noticeable brush-stroke images. An iterative coarse-to-fine rendering algorithm is developed for painting regions of varying color frequencies. Improvements over the existing technique are discussed with several examples

Table of Contents

Acknowledgments

I would like to thank my supervisors Dr. Mukundan and Dr. Tim Bell, for their patience, guidance, encouragement and everything I have learned from them which helped me to accomplish this thesis and led me through the difficulties in this research. I am thankful to Walter Raymond for reading drafts of this document. Special thanks to Roger Boyce for his useful comments about our work from an artistic point view.

In addition, thanks to my friends for all the support, encouragement and enthusiasm. Above all I would like to thank my parents and family both in New Zealand and Jordan for their encouragement, understanding and unconditional love which have helped me all through my life. Without their help this thesis would not have become a reality.

Chapter I

Introduction

Over the years painting has become one of the most important forms of art, in which painters require a great amount of skill, creativity, time and resources to create the work. The evolution of computer graphics and new technologies has enabled the creation of digital paintings automatically.

Initially, research in computer graphics has focused on photorealistic rendering of scenes, where the aim is to generate images that resembles photographs [43, 23, 14]. Recently, research began to focus on Non-Photorealistic Rendering (NPR), where the aim is to generate images that look hand painted or hand drawn [13]. This field is inspired by various artistic styles, on the other hand it is not intended to replace human creation of a painting.

The research presented in this thesis describes new ways for automatic generation of digital paintings based on previously published work in this field.

1.1 Motivation

Non-Photorealistic Rendering is commonly used in drawing programs and media applications. It requires a complex process and algorithms to (1) identify relevant image features, (2) carry out efficient image processing for

stroke placement, and (3) map the stroke images for stylized rendering. It is a field of active research with recent developments moving towards the use of programmable GPUs and computer vision algorithms.

Using computer vision algorithms to extract image features is particularly useful for the replacement of localized intensity distributions with brush-stroke images, and determining the overall image characteristics.

1.2 Objectives

This research aims to analyze and study the use of image moment feature representation capabilities for painterly-rendering algorithms, then develop moment based painterly-rendering algorithms for obtaining appropriate brush-stroke characteristics, including geometrical characteristics for stroke placement, and finally improve the efficiency of the painterly-rendering algorithms.

The major focus is to present a painterly-rendering method that aims to improve upon the previous work presented by Shiraishi and Yamaguchi [36]. Shiraishi's method first computes the brush-stroke locations from the source image, and then the stroke parameters such as the location, length, width, and angle are calculated using image moment functions. Strokes are then painted using alpha blending in order from largest to smallest.

Instead of using small image regions on locations identified by the dither function, our proposed method [31] extracts the connected color components from the image to identify the shape of larger brush-strokes, and then the stroke parameters are computed using image moment functions. This could greatly enhance the painterly appearance of the image and the rendering style. Moreover, performance improvements can be achieved by eliminating

the need for the intermediate images computed in Shiraishi's method.

1.3 Thesis Overview

The rest of this thesis is organized as follows.

- **Chapter 2** gives a brief background of Non-Photorealistic Rendering and describes the use of geometric moment functions as shape descriptors.

- **Chapter 3** reviews the moment-based painterly-rendering algorithm method presented by Shiraishi and Yamaguchi [36] and outlines limitations to the presented algorithm.

- **Chapter 4** describes the development of moment based painterly-rendering algorithms for further enhancing the expressiveness of the rendering style.

- **Chapter 5** presents and analyses the results of the research. It concludes the thesis and outlines some future directions in the area.

Chapter II

Background

In this chapter, a brief background of Non-Photorealistic Rendering techniques is reviewed, followed by a description of image moment functions that subsequent chapters will be based on.

Section 2.1 gives a brief history of NPR and gives an overview of the object-space and image-space rendering techniques. Section 2.1.2 describes the use of geometric moment functions as shape descriptors.

2.1 Overview of Non-Photorealisitic Rendering

Historically, NPR research started with 2D interactive paint systems, in which the first computer generated rendering was done by Haeberli in 1990 [15]. More recently, the evolution of NPR led to the development of 3D techniques. Teece [42] provided a summary of the historical emergence of NPR research as shown in Figure 2.1.

In this section, common NPR techniques are explained. Section 2.1.1 gives a brief overview of the object-space (3D) rendering techniques. Section 2.2 reviews some of the relevant literatures of the image-space (2D) rendering techniques.

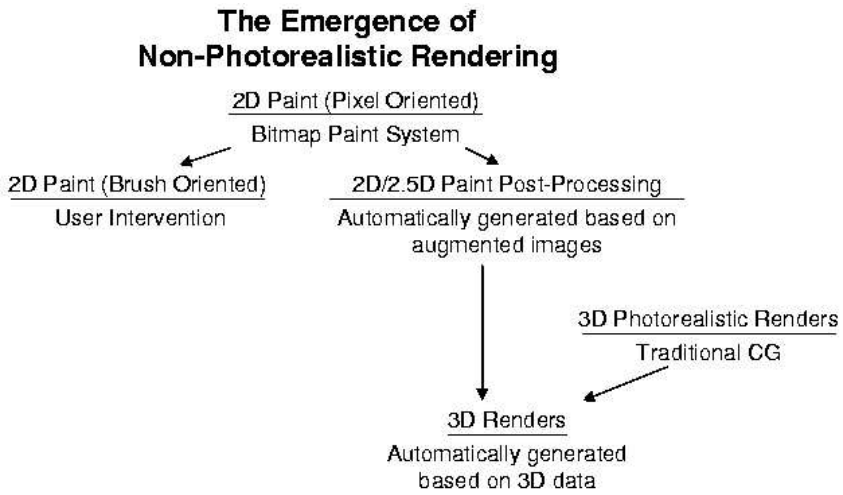

Figure 2.1: The emergence of Non-Photorealistic research. Daniel Teece, Three-Dimensional Interactive NPR [42]

2.1.1 Object-Space

Object-space techniques take as an input three-dimensional data of a scene or an object geometry to produce their NPR output. A number of successful techniques have been presented by several researchers: Hetzmann and Zorin [20], who described an algorithm for line art rendering of smooth surfaces; Winkenbach and Salesin [47], described a method for producing pen and ink renderings of parametric surfaces; Lum and Ma [27], presented a watercolor method for rendering of surfaces; Wilson and Ma [46], described a method for pen and ink illustration of complex geometry; Meier [28], developed a painterly-rendering techniques for animation using 3D model input; while Gooch et al. [11] introduced a non-photorealistic lighting model that uses tone-base shading for illumination. Many object space painterly-rendering techniques use brush-strokes to generate the painted look of the 3D model, in which strokes are associated with the corresponding geometry of the 3D

model, and then mapped to the image plane define by the virtual camera [28].

Also a number of object space techniques use silhouette algorithms [16] for complex models and scene rendering. Silhouette is widely used in NPR application as it is the simplest form of line drawing. Dooley and Cohen [8], described a system based on the techniques of traditional illustrations for automatic generation of complex three dimensional models. Gooch et al. [12] developed a system for the display of technical illustrations. Lake et al. [24] presented real-time methods to emulate pencil sketching and cartoon styles.

The main advantage of object-space NPR techniques is that they give access to the 3D data and viewing information of a scene. They produce illustrations not only coving [1] the toon texture of the surfaces in the scene, but they can also convey the 3D forms of the surfaces [35]. Having the ability to access the 3D information, programmable GPUs can introduce new NPR techniques and innovations. GPUs provide a great ability of data visualization and interactive geometry creation. Agrawala et al. [2] presented an interactive methods for creating multi-projection images and animations. They used the ability of hardware rendering tools to achieve the interactive rendering rates. Figures 2.2-2.5 show some output examples of object space techniques.

[1] Curved junction between surfaces

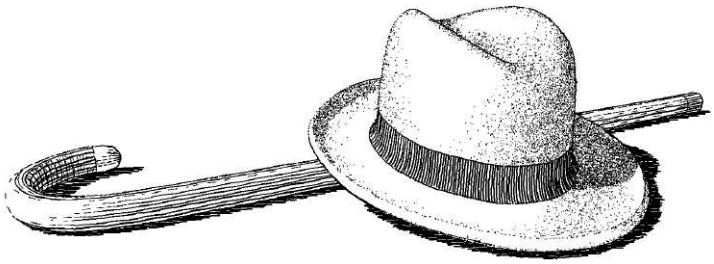

Figure 2.2: Winkenbach and Salesin's creation of a pen and ink rendering [47]

Figure 2.3: Lum and Ma's watercolor rendering [27]

Figure 2.4: Hertzmann and Zorin pencil illustration [20]

2.1.2 Image-Space

Image-space techniques produce their illustrations using two-dimensional data as an input. A good collection of NPR algorithms for illustrating different

Figure 2.5: Meier's creation of painterly-rendering using three-dimensional input [28]

artistic styles have been presented by several authors, including oil-color paintings by Hertzmann [17], watercolor paintings by Curtis et al. [7], pen-and-ink illustrations by Salisbury et al. [35], pencil illustrations by Sousa and Buchanan [38], charcoal illustrations by Bleser et al. [5], and engraving by Ostromoukhov [32].

Often image-space techniques emulate paintings (painterly-rendering) using algorithms that produce a set of brush-strokes to be mapped to the painting canvas. The new painterly-rendering technique described in this work is based upon previously published work in the field of automatic painterly-rendering algorithms, in which a painting is created using a set of brush-strokes mapped to the painting canvas. The next section outlines the major brush-stroke based painterly-renderings algorithm developed recently.

Brush-Stroke-Based Painterly-Rendering

Haeberli [15] was the first to introduce an artistic rendering system. He described an interactive painterly-rendering system and provided the user with a wide range of options and tools. The system allows the user to place

brush-strokes manually on a canvas. Each brush-stroke is described by its location, color, size, direction and shape. Haeberli also proposed an automatic way of controlling the brush-stroke orientation by using the gradient data of the source image. Figure 2.6 shows an example of Haeberli's creation of a painting [40].

Figure 2.6: Haeberli's creation of a painting [15]

Subsequent systems vary slightly by the way stroke attributes are chosen and the artistic style produced. They have been developed upon the basic idea of Haeberli, exploring new painterly-rendering algorithms with less user interaction, enhancing the expressiveness of strokes and using different inputs for video and animation rendering. Some of the related systems are further explained here [40].

Laitwinowicz [26] presented a technique that transforms ordinary video sequence into a painterly-rendered animation. The system creates a list of stroke attributes (color, position, orientation, radius, and length), in which the stroke position determines the orientation of the stroke by analysing the gradient of the image at that position. The stroke color is determined by

the input image the same way that stroke color is determined by Haeberli's method. Edge detection is used to clip strokes and seek outlines in the image to preserve the contours of the objects, where the stroke length could vary after clipping. The stroke width, on the other hand is always constant. Optical flow methods are used to achieve the coherence in the rendered video by moving strokes from frame to frame. Figure 2.7 shows a painterly-rendered image from Laitwinowicz's system [40].

Figure 2.7: A painterly-rendered image using Laitwinowicz's system [26]

Hertzmann [17] proposed a method for automatically painting brush-strokes using spline curves. Painting is done on several layers, where larger strokes are used in lower layers and thinner strokes are used in upper layers. Strokes are anti-aliased cubic B-splines, in which the intensity gradient of the image controls the orientation of the spline curves. His method was extended to processing video clips that appear painterly-rendered by painting over successive video frames in regions where the source video is changing [19, 18]. Figure 2.8 shows a painterly-rendered image from Hertzmann's technique.

Shiraishi and Yamaguchi [36] described a painterly-rendering method that

Figure 2.8: An example of painterly-rendering using curved brush-strokes using Hertzmann's technique[17]

first computes the brush-stroke locations from the source image, and then the stroke parameters such as location, length, width, and angle are calculated using image moment functions (section 2.2). Strokes are then painted using alpha blending in the order of largest to smallest. A detailed description of Shiraishi's method is given in chapter 3.

Nehab and Velho [30] extended the work of Shiraishi and Yamaguchi by using a multi-resolution technique and they introduced a parametrized stroke positions image. They also proposed a stroke placement method with the addition of a stroke normalization and dithering method for stroke positioning using variance filter and blue-noise dithering. Figure 2.9 illustrates an example of Nehab's technique..

2.2 Image Moments

Moment functions [29] are used in several image analysis applications, such as pattern recognition, image segmentation, shape descriptors, pose estimation,

Figure 2.9: A painterly-rendered image using Nehab and Valho's technique[45]

and image coding and reconstruction. Different types of moment functions exist in the image analysis field, such as geometric moments, complex moments or orthogonal moments. Each of the moment function types has its own advantages in a specific image analysis application.

For example, using complex moment functions, we can easily derive moment invariants with respect to the image rotation. On the other hand, orthogonal moment functions can represent different features of the image independently and therefore have minimum information redundancy in a set. Geometric moments are very simple, and can be easily computed from an image segment and they are widely used in pattern recognition and computer vision algorithms. A set of geometric moments defined from a two-dimensional grey-scale image generally represent the global geometrical features of an image. Recently, the feature representation capabilities of geometric moments have been used in the development of Non-Photorelistic Rendering

algorithms [36, 30, 25]. This section describes the use of the geometric moments in the Non-Photorealistic Rendering field. Section 2.2.1 defines and gives some historical background on the use of geometric moments as feature descriptors. Section 2.2.2 explains how image moments represent an image shape.

2.2.1 Geometric Moments

The first major work on image moments was published by Hu [22] in 1962. He presented a theory of two-dimensional geometric moment invariants which were used for alphabetical character recognition. Thereafter, significant contributions to the application of moment invariants have appeared. For example, moment invariants have been used in aircraft identification [9], ship identification [37], pattern recognition [3], robotic motion [10], and character recognition [4, 6, 49]. The geometric moment invariants derived by Hu have also been extended to larger sets by Wong and Siu [48] and to other moment invariant types (Teague [41], Abu-mostafa [1], Reddi [34]).

Geometric moments M_{pq} of order $(p+q)$, of an image intensity function $f(x,y)$ can generally be defined as follows:

$$M_{pq} = \int\int_{\zeta} x^p y^q f(x,y) \quad dx\, dy, \qquad\qquad p,q = 0,1,2,3,\ldots \tag{2.1}$$

where ζ denotes the pixels of an image region in the xy-plane in which the function $f(x,y)$ is defined. The monomial product $x^p y^q$ is the basis function for the moments definition in ζ.

In the case of a digital image, the double integral in equation 2.1 is replaced by a summation. The geometric moments for an $(N \times N)$ image is

given by:

$$M_{pq} = \sum_{x}^{N} \sum_{y}^{N} x^p y^q I(x, y) \tag{2.2}$$

where N is the size of the image and $I(x, y)$ is the image intensity function.

2.2.2 Shape Features Using Moments

Different orders of geometric moments describe different shape features of an image intensity distribution. The image moment of the p_{th} degree about the $x-$axis and the q_{th} degree about the $y-$axis is defined in equation 2.2. The following is a list of some image moments and how they describe image features:

- Moment of the zeroth order (M_{00}) represents the total intensity of the image intensity distribution. If the image contains an object, M_{00} represents the area of the object.

- First order moments M_{10}, M_{01} represent the intensity moments of an image about its $y-$axis, and $x-$axis respectively. Moments M_{10} and M_{01} combined with M_{00} represent the intensity centroid (x_c, y_c) as shown in equation 2.3.

- The second order moments M_{20}, M_{02}, and M_{11} represent the orientation of the image as shown in equation 2.4.

The principal shape features (Figure 2.10) are computed as follows [29, 10]:

$$x_c = \frac{M_{10}}{M_{00}};\ y_c = \frac{M_{01}}{M_{00}} \tag{2.3}$$

$$\theta = \frac{\tan^{-1}(\frac{b}{a-c})}{2} \tag{2.4}$$

$$w = \sqrt{6(a+c-\Delta)} \tag{2.5}$$

$$l = \sqrt{6(a+c+\Delta)} \tag{2.6}$$

where a, b, c and Δ are defined as,

$$la = \frac{M_{20}}{M_{00}} - x_c^2 \tag{2.7}$$

$$b = 2(\frac{M_{11}}{M_{00}} - x_c y_c) \tag{2.8}$$

$$c = \frac{M_{20}}{M_{00}} - y_c^2 \tag{2.9}$$

$$\Delta = \sqrt{b^2 + (a-c)^2} \tag{2.10}$$

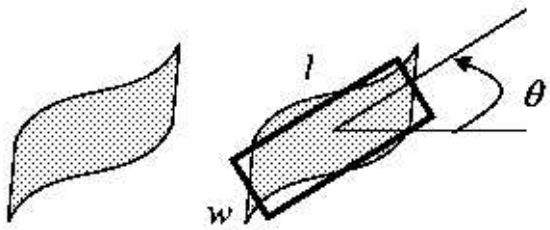

Figure 2.10: A shape and its equivalent rectangle

Chapter III

Moment-Based Painterly-Rendering — Shiraishi and Yamaguchi's Algorithm

The new painterly-rendering method described in this work is based on the previous work of Shiraishi and Yamaguchi [36]. Their algorithm is an image-based method that takes a two-dimensional color image as its input and automatically generates a two-dimensional painterly-rendered color image as its output. The algorithm approximates local regions of the source image to generate rectangular brush-strokes. Brush-stroke properties are computed using image moments (section 2.2) for small image segments of the input image. The distribution of brush-strokes is computed using a dithering algorithm without clustering, based on space-filling curves [45].

The painterly-rendering process proceeds in two phases: a preparation phase and a composition phase. Three steps are involved in the preparation phase. The first step is to define a stroke distribution over the canvas. Once a stroke distribution has been found, attributes for individual strokes can be computed. Finally, the strokes are listed from largest to smallest. In the composition phase, the strokes are painted over the canvas using alpha blending. Painting the strokes requires scaling, rotating and applying the appropriate color to the stroke template image.

This chapter reviews the moment-based painterly-rendering algorithm method presented by Shiraishi and Yamaguchi [36].

3.1 Image Segment Approximation

This section discusses how the geometric moments (section 2.2) can be used for the approximation of a local region in the source image for a single brush-stroke. The aim of this approach is to determine the stroke parameters (size, orientation and location) based on the color of the stroke and a local region of the source image.

Stroke parameters are determined in two steps. In the first step, an intensity distribution image between a region and the center color of the that region is computed. The second step computes the image moments of the intensity distribution image to determine the equivalent rectangle parameters (stroke parameters). Section 3.1.1 explains the first step and section 3.1.2 describes the second step.

3.1.1 Intensity Distribution Image

The intensity distribution image is grey scale image obtained by taking the differences of intensity values between the center pixel C and the neighboring pixels of a region C_r in the source image. The intensity distribution image is denoted by $I : R^2 \rightarrow R$ [36]. $I(x, y)$ represents the intensity value at pixel (x, y) and it is given by the following equation:

$$I(x, y) = F(d(C, C_r(x, y))) \tag{3.1}$$

where, $d(C, C_r(x, y))$ is given by the Euclidean distance between two colors C and $C_r(x, y)$ in the LUV color space. The function $F(d) \in R$ maps the color distance d to the corresponding intensity value of the intensity distribution image. $F(d)$ is given by,

$$F(d) = \begin{cases} (1 - (d - d_0)^2)^2 & \text{if } d \in [0, d_0] \\ 0 & \text{if } d \in [d_0, \infty), \end{cases} \tag{3.2}$$

where, d_0 is a threshold value representing the color distance range of color similarity. In our implementation $d_0 = 150$ is the default value. Figure 3.1 shows an example of an image region. Applying equation 3.1 to the image region in Figure 3.1, an intensity distribution image is generated as shown in Figure 3.2.

Figure 3.1: An image region

3.1.2 Equivalent Rectangle Parameters

The stroke parameters are determined by computing the image moments of the intensity distribution image. The result is the corresponding equivalent rectangular shape that approximates the stroke region. Figure 3.3 shows the

Figure 3.2: An intensity distribution image

corresponding equivalent rectangle of the image in Figure 3.1. The center coordinate (x_c, y_c), the angle of orientation θ between the main axis of the stroke and the $x-$axis of the image, width w and length l of equivalent rectangle are computed using equations 2.3, 2.4, 2.5 and 2.6 respectively.

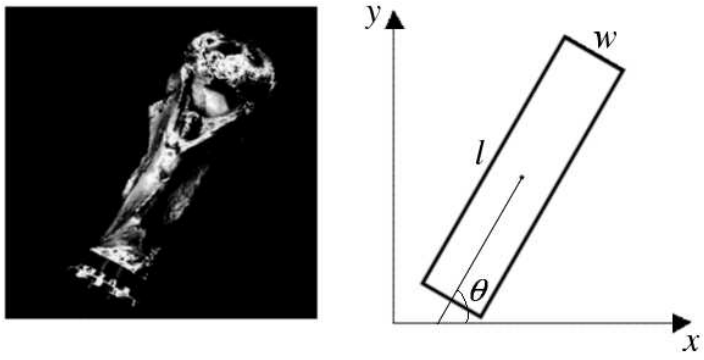

Figure 3.3: An image and its corresponding equivalent rectangle

In the context of painterly-rendering, the computed parameters are used to map a brush-stroke template image into the equivalent rectangle for an image segment.

3.2 The Painterly-Rendering Algorithm

Using the concepts outlined in the previous section, the painterly-rendering algorithm proceeds as shown in Figure 3.4. Image moments are used both to obtain image features and to determine the stroke distribution. The following sections (3.2.1-3.2.4) explain how the painterly-rendering algorithm works in detail.

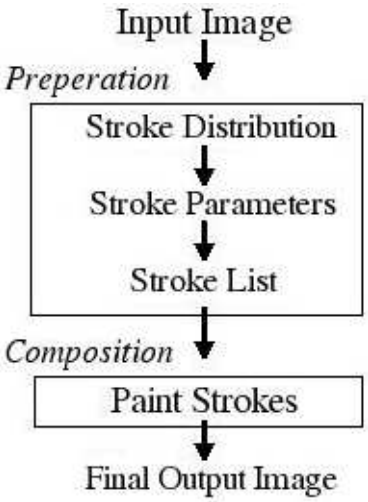

Figure 3.4: The painterly-rendering process

3.2.1 Stroke Distribution

The stroke distribution depends on the level of detail (variation in intensity values) in a region. It is obtained in two steps: estimation of stroke area intensity and generation of stroke locations.

The stroke area intensity estimation is a measure of variations in intensity values surrounding a pixel and can be obtained by taking the sum of squared differences of intensity values between the pixel and its neighboring pixels in

an image region. If k donates half of the region's size, we can obtain this measure as the value of $g(i,j)$ at pixel (i,j) in the following equation:

$$g(i,j) = \sum_{u=i-k}^{i+k} \sum_{v=j-k}^{j+k} I(u,v) \tag{3.3}$$

where, I is equation 3.1 and its center color C is at pixel (i,j).

If $g(i,j)$ is inverted and normalized to a valid intensity range, then a low value of $g(i,j)$ indicates a large variation of intensity in the neighborhood of (i,j). The image obtained by assigning this value of $g(i,j)$ to pixel (i,j) is called the stroke area image. Figure 3.6 shows a stroke area image for the the test image in Figure 3.5.

Figure 3.5: A test image

The stroke area image is used to generate the stroke location image by applying a dithering algorithm without clustering. The dithering algorithm distributes stroke locations over an image, where each location is represented by a dot. In the dithered image, regions of high detail of the source image will correspond to dark regions with high density of dots. The original im-

Figure 3.6: A stroke area image

plementation [36] uses a modified version of a space-filling curve dithering algorithm [45] to generate the stroke locations image. However, our implementation uses the Hilbert dithering algorithm[1] [45] to generate the stroke locations. Figure 3.7 shows the stroke locations image generated from the stroke area image in Figure 3.6.

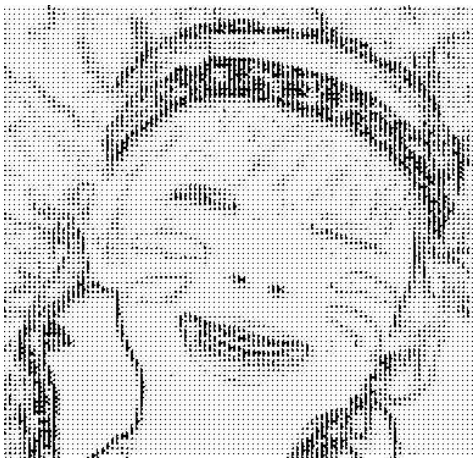

Figure 3.7: A stroke locations image

[1] See appendix A for more detail on dithering algorithms

3.2.2 Stroke Parameters

Stroke parameters are computed for each stroke location (dot) in the stroke locations image. Stroke parameters include color C, location (x, y), orientation θ and size (width w and length l). Four steps are involved in the computation of the stroke parameters, as follows:

1. Color definition: The color C of the stroke is determined by sampling the color of the input image at the stroke location.

2. Input image cropping: Using a user specified maximum stroke size parameter s, an $s \times s$ image is cropped from the input image. The center of the cropped image is the stroke location determined from the stroke locations image.

3. Intensity distribution image generation: As explained in section 3.1.1, the intensity distribution image is computed using the cropped image and the computed stroke color C.

4. Equivalent rectangle computation: Using the computed intensity distribution image, the equivalent rectangle of the cropped image is computed as explained in section 3.1.2. The equivalent rectangle parameters ((x_c, y_c), θ, w and l) are assigned as the stroke parameters, where the location of the stroke is moved to the center of the equivalent rectangle.

3.2.3 Stroke List

A stroke list is generated by sorting the strokes according to their area size ($w \times l$) in order from largest to smallest. It is imperative to list the strokes in this order to preserve the details of the input image, so that smaller strokes will not get obscured by larger strokes in the painting process.

3.2.4 Painting Strokes

The strokes are painted one after the other in the order of the stroke list. Painting the strokes is performed using alpha blending, where each stroke is painted on the canvas according to its set parameters ((x_c, y_c), θ, w and l). This process requires the use of the appropriate transformations (scaling and rotating) and color of the stroke template image before blending it on the painting canvas. Figure 3.8 shows the painterly-rendered version of the image in Figure 3.5.

3.2.5 Limitations

The primary limitation of the method described above is that large regions of nearly constant color are not painted by correspondingly large brush-strokes. The dither function imposes a constraint on the maximum distance between two stroke locations, which in turn limits the size of the brush-strokes used. It may also be noted that a large region of nearly white color in the stroke area image represents only a collection of regions of similar color, and not necessarily a single region of constant color. The implementation of the painterly-rendering method described was tested with several images, and the following limitations were found:

Figure 3.8: The painterly-rendered image of Figure 3.5

- Brush-Strokes are always mapped into a small fixed sized rectangular shape, and therefore the important characteristics of the brush-stroke are not represented or made visible in the final image , as shown in the cut-out image given in Figure 3.8.

- Finer details are also not represented due to the fixed size of the brush-stroke.

- The performance of the painterly-rendering algorithm is slow due to the creation of the intermediate images (a stroke area image and a stroke position image) before the final output image.

The painterly appearance of an image can be further enhanced if regions of similar color can be identified and grouped as a single component and then used for mapping brush-strokes. This is the core idea of the newly developed algorithm described in chapter 4.

Chapter IV

Moment-Based Painterly-Rendering — New Algorithms

The previous computer-generated moment based algorithm [36] has shown the lack of an important aspect in regards to the painterly appearance of a painted image. The fact that large regions of nearly constant color are not represented with correspondingly large brush-strokes imposes a limitation on the painted look of an image. Therefore, for a more stylized rendering of images, it is imperative to have large brush-strokes visible in regions of nearly constant color. This requirement has motivated us to consider the use of connected color components as an important feature for painterly-rendering. In other words, the painterly appearance of an image can be further enhanced if regions of similar color can be identified and grouped as a single component and then used for mapping brush-stroke images.

This chapter describes the development of moment based painterly-rendering algorithms for further enhancing the expressiveness of the rendering style. Section 4.1 explains how connected color components are computed using a recursive algorithm. Section 4.2 shows how inverse transformation is used for mapping a brush image. Section 4.3 describes a layered approach for moment-based painterly-rendering, using connected color components. Section 4.4 explains a progressive moment-based painterly-rendering technique

using connected color components.

4.1 Connected Color Component

To achieve the painterly appearance of large brush-strokes in an image, similar color regions are computed by using a recursive connected color component algorithm. The connected color components of an image are computed as follows:

1. **Initialize -** The algorithm starts at the color of pixel $(0, 0)$ of the input image.

2. **Propagate -** Recursively propagate using 4-point connected colored pixels (see Figure 4.1).

3. **Test -** Check if the current pixel color belongs to the region.

4. **Update -** Update the flag table for scanned pixels (by changing its default value)

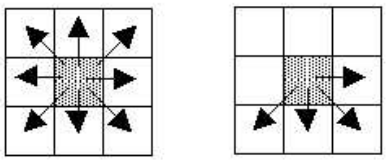

Figure 4.1: 8-point and 4-point connectivity

The connected color algorithm [21] recursively propagates across the image starting with the color of the pixel $(0, 0)$, testing each of the neighboring unscanned pixels for color similarity. An array buffer is used to keep track

of the scanned and unscanned pixels. If the scanned pixel's color is less than a given color similarity threshold, the pixel is colored with the color of the connected region and the value of the buffer array is updated.

The threshold value controls the size of the connected color regions. Increasing the color similarity percentage increases the range of color values that can be grouped under one color region. The program code below shows the corresponding implementation of a recursive 4-point connected color algorithm.

```
void component(int x, int y, RGB clr){
  float colorDistance;
  RGB color;
  if(flag[x][y]>0) return; // (x, y) is already scanned
  if(x>width || y>height || x < 0 || y < 0) return;
  inputImage.getcolor(x, y, color);
  colorDistance=(color.r-clr.r)^2+(color.g-clr.g)^2+(color.b-clr.b)^2;
  if(colorDistance<colorThreshold){
    connectedImage.setcolor(x,y,clr);
    componentCounter++; // counter for the number of connected pixels
    flag[x][y]=flagValue; // update the buffer
    component(x+1,y,clr);
    component(x+1,y-1,clr);
    component(x,y-1,clr);
    component(x-1,y-1,clr); }
}
```

Even though a recursive 8-point connected component algorithm is the simplest to implement, it may require a large stack space for images of size larger than 100×100 pixels. As the image is scanned from top to bottom and left to right, neighboring pixels on the top-right, top, top-left, and left of the current pixel would have already been processed. We could therefore minimize the amount of computation by comparing only the remaining four neighbors of the current pixel, as shown in Figure 4.1.

Since connected components are used only for mapping brush-stroke images, we can also use a simple one-pass iterative connected component algorithm with 4-point connectivity as shown in Figure 4.1. With this process, connected components of certain shapes, as shown in Figure 4.2, will get split into two components, but this is acceptable since such an image would require brush-strokes in more than one direction anyway.

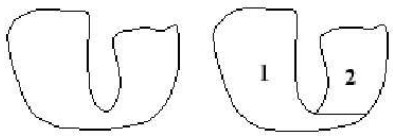

Figure 4.2: Splitting of connected components

4.2 Inverse Transformation

An image can be mapped onto a canvas by applying a combination of simple affine transformations (rotation, translation, or scaling). The methods described in this chapter use an inverse transformation to map the brush-stroke image into a canvas with the appropriate color of the stroke.

Figure 4.3 shows an outline of a stroke image with length L and width W. To map a point (x, y) in an equivalent rectangle with the parameters

$((x_c, y_c), l, w$ and $\theta)$ to axis-aligned brush image coordinates $(x^{''}, y^{''})$, the equivalent rectangle is rotated by $(-\theta)$ about (x_c, y_c) to obtain $(x^{'}, y^{'})$ as shown in Figure 4.4. The inverse transformation that maps point (x, y) to the brush image coordinates $(x^{''}, y^{''})$ is given below:

$$
\begin{aligned}
x^{'} &= (x - x_c)cos(\theta) + (y - y_c)sin(\theta) \\
y^{'} &= -(x - x_c)sin(\theta) + (y - y_c)cos(\theta) \\
x^{''} &= (\tfrac{x^{'}}{l} + \tfrac{1}{2})L \\
y^{''} &= (\tfrac{y^{'}}{w} + \tfrac{1}{2})W
\end{aligned}
\tag{4.1}
$$

After obtaining the coordinates $(x^{''}, y^{''})$, the color value at the image pixel (x, y) is set to that of the brush image at $(x^{“}, y^{“})$.

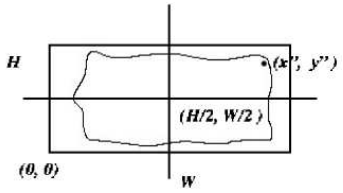

Figure 4.3: An outline of a brush-stroke image

4.3 Method One: Layered Approach

The ideas outlined in sections 4.1 and section 4.2 can be implemented in a moment-based painterly-rendering algorithm for further enhancing the painterly appearance of a rendered image. Regions where large brush-strokes are to be

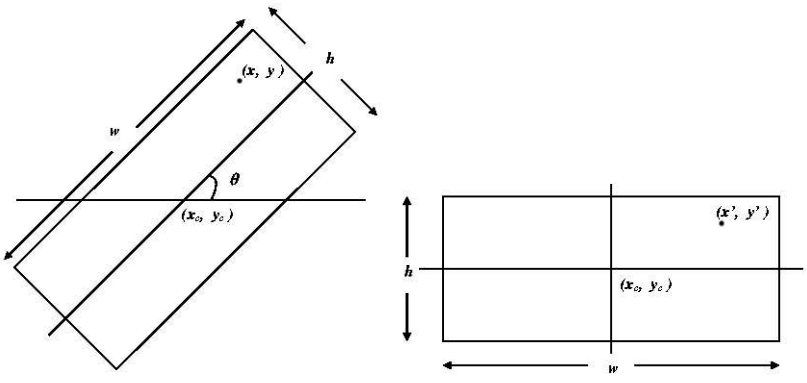

Figure 4.4: Brush image mapped to connected components

used are first identified by extracting the connected components, before mapping the brush-stroke template image. This process is illustrated in Figure 4.5.

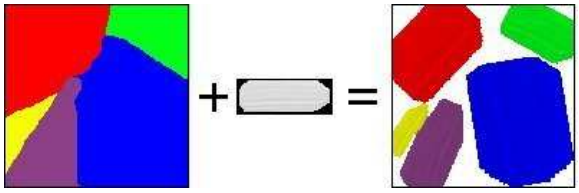

Figure 4.5: Brush image mapped to connected components

The painterly-rendering foundation described in this section consists of three layers: (1) a layer of large painted brush-strokes, (2) a rough painterly-rendered layer, (3) and a layer to highlight the edges of an image. The three layers are blended together to produce the final painterly-rendered image. Figure 4.6 shows the process of painterly-rendering, employing a layered approach. Sections 4.3.1, 4.3.2 and 4.3.3 explain the three layers respectively.

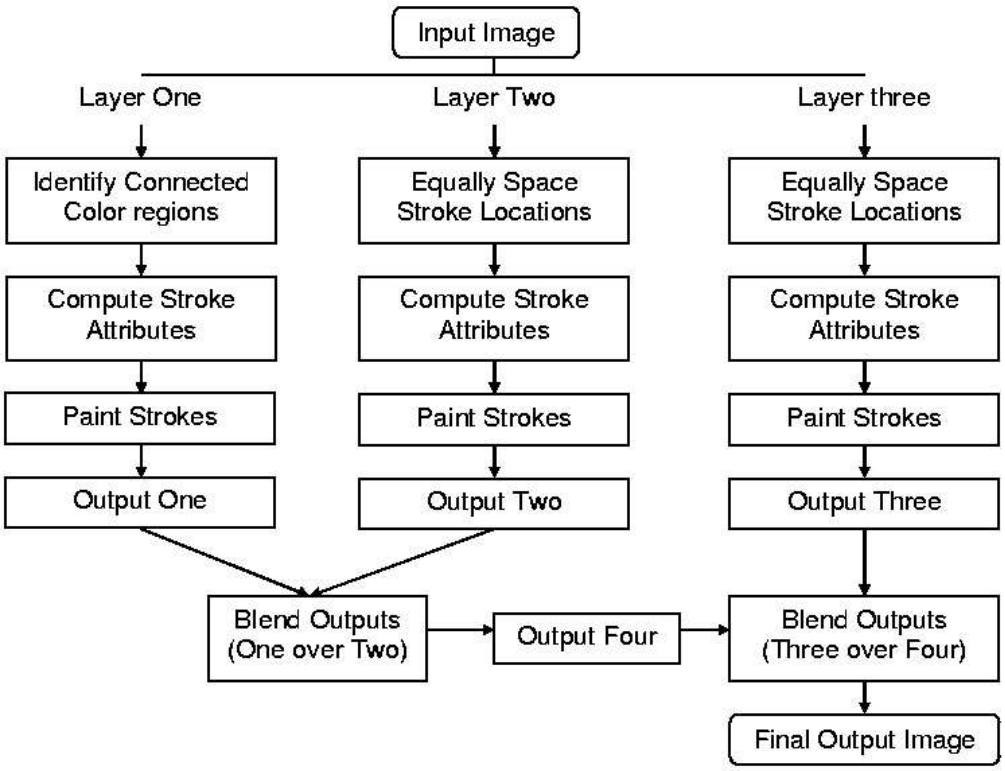

Figure 4.6: The process of a painterly-rendering algorithm

4.3.1 Layer One

Layer one introduces the use of large brush-strokes in the painterly-rendered image. Large brush-stroke areas are identified as explained in section 4.1, in which similar color regions are identified according to a color similarity threshold value. To avoid obvious large brush-strokes across the painterly-rendered image, connected regions larger than a given threshold are divided into several connected color regions.

Each connected color region will be considered as a brush-stroke and its equivalent rectangle attributes ((x_c, y_c), θ, and (w,l)) are determined based on the image moments of the connected region, as explained in section 2.2.2.

Strokes are then painted on a blank canvas by applying an inverse transformation to the brush-stroke template image with the appropriate stroke color, as explained in section 4.2. Figure 4.7 shows the output image produced from layer one.

Figure 4.7: The output image of layer one representing large brush-strokes

4.3.2 Layer Two

Layer two represent a rough painterly-rendered output of the input image. This layer is obtained by painting fixed size brush-strokes across a blank canvas using an equidistant stroke placement technique. Stroke locations are determined by using an equally spaced stroke locations image, in which each location is represented by a dot in the image, where dots are 5 pixels apart. Using an equidistant stroke placement technique eliminates the need for the expensive performance computation of the dither image used by Shiraishi and Yamaguchi [36]. Figure 4.8 shows an example of an equally spaced stroke locations image.

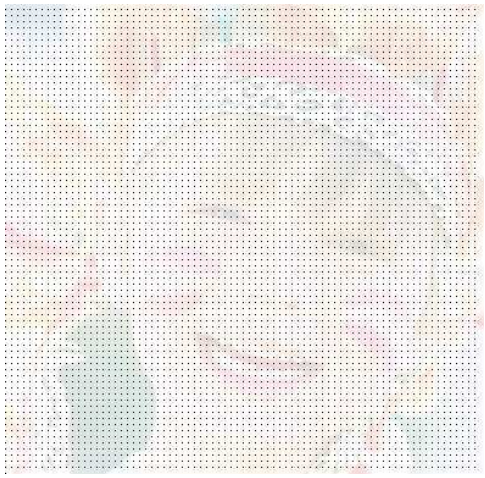

Figure 4.8: Equally spaced stroke locations image

The stroke maximum size is set to 10×10 pixels; this size will allow the painted strokes to cover the painting canvas with no painting cracks. Painting the strokes on a blank canvas requires the computation of the stroke parameters. Shiraishi and Yamaguchi's [36] (section 3.2.2) method for computing stroke parameters is used in our algorithm to determine the stroke parameters for a specified stroke location. Each stroke is painted on the canvas by applying an inverse transformation (section 4.2) to the brush-stroke template image with the appropriate color of the stroke. This process is preformed using the traditional alpha blending as the strokes will overlap to give a roughly painterly-rendered output image. Figure 4.9 shows the output image of layer two after painting the stroke.

4.3.3 Layer Three

This layer gives a painted effect for the sharp edges in the image. To detect edges in the image, the Sobel edge detection algorithm is used [44]. The edge detection algorithm uses a pair of 3×3 convolution masks applied to

Figure 4.9: The output of layer two from the image shown in Figure 3.5

each pixel in the image, in which the intensity value I of the pixel is used. One mask is used to estimate the gradient in the horizontal direction (G_x) and the other mask estimates the gradient in the vertical direction (G_y). Figure 4.10 shows convolution masks for the Sobel edge detector used.

-1	0	+1
-2	0	+2
-1	0	+1

a

+1	+2	+1
0	0	0
-1	-2	-2

b

Figure 4.10: Sobel convolution masks (a) horizontally G_x (b) vertically G_y

The values of G_x and G_y are computed using the following equations:

$$\begin{aligned} G_x = {} & I(x+1, y+1) + 2 \times I(x+1, y) + I(x+1, y-1) - \\ & I(x-1, y+1) - 2 \times I(x-1, y) - I(x-1, y-1) \end{aligned} \tag{4.2}$$

$$\begin{aligned} G_y = {} & I(x-1, y+1) + 2 \times I(x, y+1) + I(x-1, y+1) - \\ & I(x-1, y-1) + 2 \times I(x, y-1) + I(x-1, y-1) \end{aligned}$$

The absolute magnitude of the gradient at each pixel is given by equation 4.3 below:

$$| G |= \sqrt{(G_x)^2 + (G_y)^2} \tag{4.3}$$

If the the magnitude is greater than a specified threshold, the pixel is marked in black as an edge. Otherwise, the pixel is ignored. Figure 4.11 shows the detected edges from Figure 3.5.

Figure 4.11: Edges detected from Figure 3.5 using the Sobel edge detector

In the context of our algorithm, each black point along the detected edge represents a brush-stroke location. Applying a small stroke for each stroke location of the detected edge, will give a fine painted effect for the edges in the painterly-rendered image. The stroke's maximum size is set to 5×5 pixels for each stroke location, for which the stroke parameters are computed as explained in section 3.2.2. Painting the strokes is performed by mapping the brush-stroke template image using an inverse transformation as explained in section 4.2. Figure 4.3.3 shows the output image of layer three.

Figure 4.12: The output of layer three from the image shown in Figure 3.5

4.3.4 Blending Output Images

To produce the final painterly-rendered image and preserve the details of the larger strokes in the final image, the output image of layer one (Figure 4.7) is blended over the output of layer two (Figure 4.9). Finally, the resulting image will have the output of layer three (Figure 4.12) blended over it to produce the painterly-rendered image. Figure 4.13 shows a painterly-rendered image of the input image in Figure 3.5.

4.3.5 Summary

The result of the process in Figure 4.6 gives a more stylized rendering of images and also enhances the performance of the rendering algorithm slightly. However, the presented approach can lead to some noticeable artifacts and image details can be lost due to the way the second layer is constructed. Some limitations of the presented approach are as follows:

- Much detail of the input image can be lost due to the second layer.

Figure 4.13: The painterly-rendered version of the image in Figure 3.5

- The creation of three layers can slow the performance of the algorithm.

- The creation of the third layer to enhance painted edges, imposes preformance issues as a result of brush strokes being mapped to each black point along the detected edges.

- In some cases regions of high frequency color variations can not be handled.

4.4 Method Two: Progressive Approach

The previous sections (3.2 and 4.3) have provided the context and the goal for the painterly-rendering algorithm described in this section. The algorithm

employs a progressive rendering approach, in which strokes are painted on a blank canvas iteratively to achieve the final painting output.

The algorithm consists of three main steps: (1) identifying connected color regions from the input image repeatedly, (2) computing the stroke attributes from connected regions, and (3) painting the brush-strokes on a painting canvas. Figure 4.14 illustrates the process of producing the painterly-rendered output image.

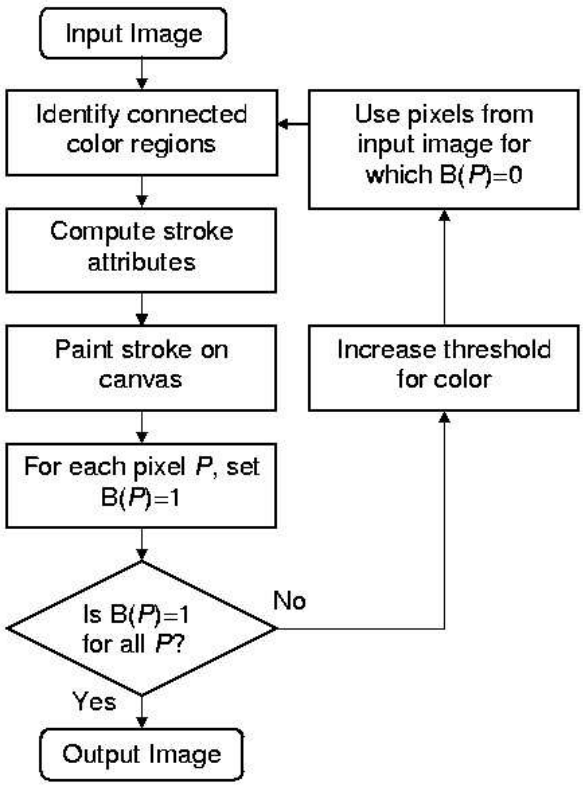

Figure 4.14: Flow chart for the modified painterly-rendering algorithm

The process is further explained below.

4.4.1 The Painting Process

An artist usually produces paintings by applying large brush-strokes for constant color areas, and finer strokes for detailed areas in the image. The process (Figure 4.14) uses different brush-stroke sizes depending on the size of the connected color region.

The algorithm starts with a blank canvas and a bit array $B(P)$ of the same size as the input image. The array which is initialized to zero, is used to check if a pixel P has already been painted or not. Connected regions of similar color are then identified as explained in section 4.1. Regions whose size is smaller than a given threshold are ignored (leaving gaps as shown in Figure 4.15), and later merged into the surrounding regions by increasing the color similarity threshold. This will cause regions of large variations to appear smudged, giving them a painterly appearance. Similarly, to avoid obvious large brush-strokes across the painterly-rendered image, connected regions larger than a given threshold are divided into several connected color regions.

Brush-stroke parameters ($(x_c, y_c), l, w$ and θ) are computed from geometric moments (section 2.2.2) for each connected color region. According to the computed parameters, each stroke is painted on canvas by applying an inverse transformation to the brush-stroke template image, as explained in section 4.2. The bit array buffer B is updated for each painted pixel P, i.e., $B(P) = 1$. Figure 4.15 shows the brush images applied to the equivalent rectangles of connected components obtained from Figure 3.5.

As seen in Figure 4.15, the initial painting of large brush-strokes partially covers the painting canvas. Running the process iteratively for unpainted

Figure 4.15: The initial connected brush-strokes for the image in Figure 3.5

regions with $B(P) = 0$, and simultaneously increasing the color similarity threshold for each iteration leads to painting the whole image with different sizes of brush-strokes. Figure 4.16 shows different stages of painting an image before getting to the final output image.

From Figure 4.16, it can be shown that the rendering algorithm used is a progressive rendering algorithm, which uses a coarse-to-fine approach for selecting stroke regions. The final painterly-rendered image of Figure 3.5 using the described algorithm is shown in Figure 4.17.

4.4.2 Summary

This section has presented a moment based painterly-rendering for artistic rendering, using progressive rendering, as shown in Figure 4.16. The progressive rendering algorithm iteratively maps strokes to the painting canvas using a coarse-to-fine approach.

The result obtained from the process shown in Figure 4.17, significantly improves the quality of stylized rendering of the painted images. Large brush-

Figure 4.16: Progressively rendering the image in Figure 3.5

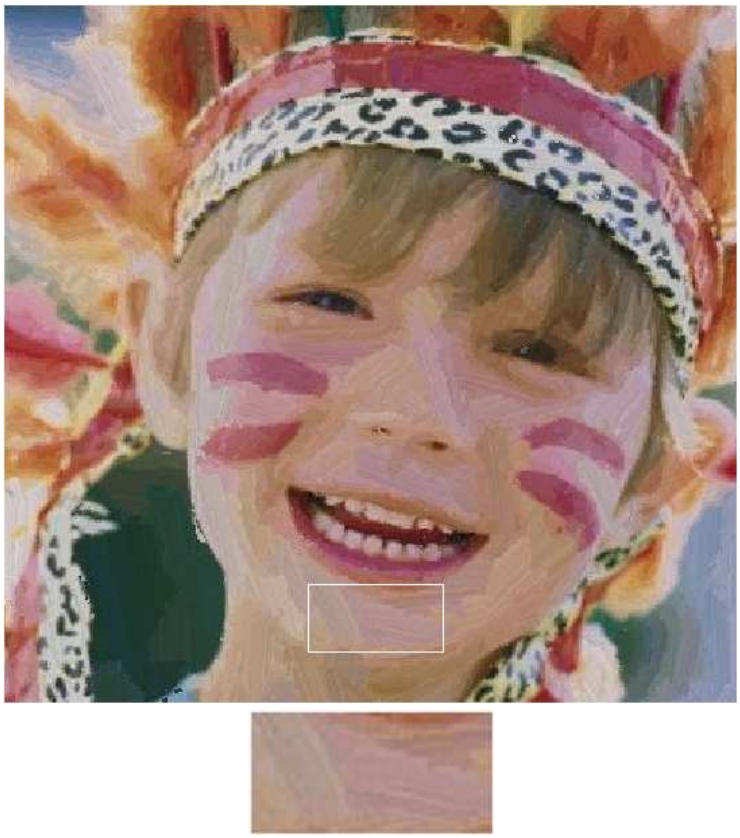

Figure 4.17: The painterly-rendered version of the image in Figure 3.5

strokes are clearly visible in regions of nearly constant color, as shown in the cut-out image of Figure 4.17. Performance improvements over existing techniques are also achieved by eliminating the need for the computation of the dither image.

Chapter V

Results and Comparative Analysis

This chapter presents and analyses the results of our research. The results are automatically generated two-dimensional painterly-rendered images from the algorithms described in chapters 3 and 4.

The implementation of the algorithms are tested with various types of images to determine the effectiveness of the painterly-rendering techniques described. While it is difficult to quantify the accuracy of the results (a painterly-rendered image does not necessarily have a quality metric), a detailed comparative analysis is preformed on the results obtained from the techniques described in this work. Results obtained from image editing software (Adobe Photoshop)[33] are also used in the analysis.

Section 5.1 presents some of the results obtained using the techniques described in chapters 3, and 4, and from image editing software. Section 5.2 gives a comparative analysis and evaluation of the results. Section 6 concludes the research work and outlines some future directions in this area.

5.1 Images

This section presents some of the resulting images[1] of the painterly-rendering techniques described in this work. The original test images used (Figure 5.4, 5.9 and 5.14) are a mixture of mainly three image properties: cluttered background, constant color regions and image regions with high frequency of color variation.

Figure 5.4 shows a test image of an African safari tree used in our experimental analysis. It contains both regions of constant color requiring large brush-strokes (applied by extracting connected components), and regions with large color variation requiring finer brush-strokes. The output images of Figure 5.4 using four different painterly-rendering techniques are shown in Figures 5.5 - 5.8.

Figure 5.9 shows a test image of a Siberian tiger. It is another image which demonstrates painterly-rendering using different sizes of brush-strokes depending on the techniques used. The output images of Figure 5.9 using four different painterly-rendering techniques are shown in Figures 5.10 - 5.13.

Finally, Figure 5.14 shows a test image of shore rocks, which contains predominantly regions of high frequency color variation, and therefore requiring many fine brush-strokes. The output images of Figure 5.14 using four different painterly-rendering techniques are shown in Figures 5.15 - 5.18.

[1] Images can be viewed online -
http://www.cosc.canterbury.ac.nz/research/PG/mho33/thesis

5.1.1 Color Similarity Threshold

The color similarity threshold value represents the color distance range of color similarity. A higher threshold value will result in a large connected color regions of similar color, where a lower threshold value will produce many smaller color regions. Painting an image with few large stroke does not give a clear painterly rendered image; furthermore, painting an image with many smaller brush strokes will slow down the process of painting an image. An artist usually uses different brush-strok sizes when rendering an image and to allow the algorithm to render images relatively similar to how an artist would render an image, the threshold value is chosen to produce a painterly rendered image that contains large, medium and small brush-strokes. Figures **??** - 5.2 shows an example of an image (3.5) painted using three different threshold values.

Figure 5.1: A high color similarity threshold painterly rendered image

Figure 5.2: A medium color similarity threshold painterly rendered image

Figure 5.3: A small color similarity threshold painterly rendered image

5.2 Analysis and Evaluation

Evaluation of painterly-rendering algorithms is one of the unsolved problems of NPR research. Hertzmann [18] described NPR evaluation as follows:

> One challenge of NPR research is that it is difficult to perform meaning-

ful scientific tests in a field where judgments are ultimately subjective.

> The difficulty of defining a scientific methodology does not make the work any less valuable, although the evaluation is ultimately somewhat subjective. Like all research, NPR requires a great deal of guess work and stumbling around in the dark; only after experience does the community gain some feeling for which ideas are valuable. One can easily find examples of useful reasoning and research areas where exact quantitative measurements are impossible (e.g. in political science), and, conversely, quackery justified by plausible scientific methodology (e.g. in political science).

To evaluate and show the effectiveness of the moment based painterly-rendering algorithms described in this work, a comparative analysis is preformed on the results shown at the end of this chapter. Each of the algorithms and result images are tested on the support of the following properties:

1. Visible **large brush-strokes** in the painterly-rendered image.

2. **Finer brush-strokes** in the painterly-rendered image, as required.

3. **Geometric extraction** using image moment shape descriptors.

4. Perform painterly-rendering on images with **cluttered background**.

5. Perform painterly-rendering on images with **high color variation**.

6. Perform painterly-rendering on images with **constant color regions**.

Table 5.1 shows the properties supported by the algorithms and output images of Shiraishi and Yamaguchi (chapter 3), moment based layered approach (section 4.3), moment based progressive approach (section 4.4) and Adobe Photoshop software [33].

	Large brush strokes	Fine brush strokes	Geometric extraction	Cluttered background	High color variation	Content color regions
Yamaguchi		√	√	√	√	√
Layered approach	√	√	√	√		√
Progressive approach	√	√	√	√	√	√
Photoshop				√	√	√

Table 5.1: painterly-rendered image properties of different techniques

From table 5.1 it can be seen that the moment-based progressive approach can handle all of the listed properties. The advantage of being able to generate larger and noticeable brush-strokes further enhances the expressiveness of the rendering style. Furthermore, the ability to handle high color variation with finer brush-strokes greatly improves the rendering style of the painted image, as seen in Figure 5.17. Adobe Photoshop lacks the use of shape descriptors for geometric extraction and generates painterly-rendered images of smudged brush-strokes, in which large and fine brush-strokes are not clearly visible.

Figure 5.4: A test image of an African safari tree

Figure 5.5: The painterly-rendering of the image shown in Figure 5.4 using Shiraishi and Yamaguchi technique (chapter 3)

Figure 5.6: The painterly-rendering of the image shown in Figure 5.4 using moment based layered technique described in section 4.3

Figure 5.7: The painterly-rendering of the image shown in Figure 5.4 using moment based progressive technique described in section 4.4

Figure 5.8: The painterly-rendering of the image shown in Figure 5.4 using Adobe Photoshop 8.0

Figure 5.9: A test image of a Siberian tiger

Figure 5.10: The painterly-rendering of the image shown in Figure 5.9 using Shiraishi and Yamaguchi technique (chapter 3)

Figure 5.11: The painterly-rendering of the image shown in Figure 5.9 using moment based layered technique described in section 4.3

Figure 5.12: The painterly-rendering of the image shown in Figure 5.9 using moment based progressive technique described in section 4.4

Figure 5.13: The painterly-rendering of the image shown in Figure 5.9 using Adobe Photoshop 8.0

Figure 5.14: A test image of shore rocks

Figure 5.15: The painterly-rendering of the image shown in Figure 5.14 using Shiraishi and Yamaguchi technique (chapter 3)

Figure 5.16: The painterly-rendering of the image shown in Figure 5.14 using moment based layered technique described in section 4.3

Figure 5.17: The painterly-rendering of the image shown in Figure 5.14 using moment based progressive technique described in section 4.4

Figure 5.18: The painterly-rendering of the image shown in Figure 5.14 using Adobe Photoshop 8.0

Chapter VI

Summary

6.1 Conclusion

This thesis[1] has presented a new approach to the moment-based algorithm described by Shiraishi and Yamaguchi for artistic rendering of images, where brush-stroke parameters are computed using geometric moments of local intensity distributions. Instead of using small windowed regions on locations identified by the dither function, our algorithm uses connected components for stroke placement. Connected components also allow the rendering of large brush-stroke images in regions of nearly constant color. With this modification, the quality of the stylized rendering of the painted images could be significantly improved. Performance improvement over existing techniques is also be achieved by eliminating the need for the computation of the stroke area image and the dither image.

6.2 Future Work

There are many possibilities for further research arising from the work presented in this thesis.

[1] A PDF copy of this thesis can be downloaded from
http://www.cosc.canterbury.ac.nz/research/PG/mho33/thesis

A few suggestions for possible future work are as follows:

- Further research in this area can be directed towards enhancing the edges between two features of different colors in an image.

- Investigate the feature representation capabilities of advance image moment functions (orthogonal moments).

- Having described an image by a set of moments, it may prove useful to investigate orthogonal moments and their image reconstruction capabilities.

6.3 Publication

The work presented in this thesis was published in the international conference of Computer Graphics, Imaging and Visualization (CGIV 2006), Australia. The paper[2] details are as follows:

- Mohammad Obaid, Ramakrishnan Mukundan, and Tim Bell. Enhancement of moment based painterly-rendering using connected components. In *CGIV 2006: Computer Graphics, Imaging and Visualization*, pages 378–383, Sydney, July 2006. IEEE Computer Society.

[2] A PDF copy of the conference paper can be downloaded from http://www.cosc.canterbury.ac.nz/research/PG/mho33/thesis

Appendix A

Dithering Algorithms

A.1 Floyd-Steinberg Algorithm

Floyd-Steinberg is a dithering algorithm [39]. The algorithm achieves dithering by distributing the approximation error of a pixel to its neighboring pixels. More precisely, the distribution of error of a pixel is computed by adding 7/16 of it to the pixel to its right, 3/16 of it to the pixel to its lower left, 5/16 to the pixel below, and 1/16 to the pixel to its lower right.

For example, if we have a matrix of pixel values:

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

then the center pixel approximation error is distributed by Floyd-Steinberg algorithm as follows:

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & \frac{7}{16} \\ \frac{3}{16} & \frac{5}{16} & \frac{1}{16} \end{pmatrix}$$

The following pseudo code presents Floyd-Steinberg dithering algorithm [39]:

```
FOR y:=height(S)-1 TO 1 DO
  FOR x:=width(S)-1 TO 1 DO
    K:=approximate(S[x,y]);
    O[x,y]:=K;
    error:=S[x,y] - K;
    S[x+1,y]:=S[x+1,y]+(7/16)*error;
    S[x-1,y-1]:=S[x-1,y-1]+(3/16)*error;
    S[x,y-1]:=S[x,y-1]+(5/16)*error;
    S[x+1,y-1]:=S[x+1,y-1]+(1/16)*error;
  END;
END;
```

where S is the input image, O is the output image and *approximate*() is a function that returns the possible pixel values to be displayed in the output image of the current pixel. Figure A.2 shows the output image of Figure A.1 being processed by Floyd-Steinberg dithering algorithm.

Figure A.1: The Lena image

Figure A.2: The output of Floyd-Steinberg dithering algorithm

A.2 Hilbert Dithering

The Hilbert space filling curve is a one dimensional curve which visits every point within a two dimensional space. Figure A.3 shows an example of Hilbert curve iteration.

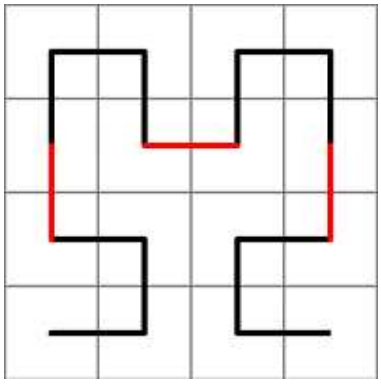

Figure A.3: Hilbert space filling curve for 16 pixels

Hilbert dithering can be achieved by scanning the image with a Hilbert space filling curve and generating a dot depending on the intensity over a region and the area of the region. The Hilbert dithering algorithm consists of three steps [45]:

1. Divide the source image into smaller regions based on the trace of the Hilbert curve.

2. Compute the average intensity value for each region.

3. Depending on the computed intensity the dot patterns are generated.

The following pseudo code presents the Hilbert dithering algorithm [45].

```
Initialize intensity accumulator;

WHILE(input image){
   Advance image pointer along the scan
   path to the end of interval;

   Move backward N pixels, accumulating
   the intensity of the input image;

   Move forwad N pixels along the path,
   setting the output pixels;

   IF(accumulator = R) THEN{
     decrement R from accumulator;
     set output pixel to black;
   }ELSE{
     set output pixel to white;
   }
}
```

Figure A.4 shows the output image of Figure A.1 being processed by the Hilbert dithering algorithm.

Figure A.4: The output of the Hilbert dithering algorithm

References

[1] Y.S. Abu-Mostafa and D. Psaltis. Recognitive aspects of moment invariants. In *IEEE Trans. Pattern Analysis and Machine Intelligence*, volume 11, pages 698–706, 1984.

[2] M. Agrawala, D. Zorin, and T. Munzner. Artistic multiprojection rendering. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pages 125–136, London, UK, 2000. Springer-Verlag.

[3] F. L Alt. Digital pattern recognition by moments. In *JACM*, pages 240–258, 1962.

[4] S.O. Belkasim, M. Shridhar, and M. Ahmadi. Pattern recognition with moment invariants: a comparative and new results. In *Pattern Recognition*, pages 1117–1138,, 1991.

[5] Teresa W. Bleser, John L. Sibert, and J. Patrick McGee. Charcoal sketching: returning control to the artist. *ACM Trans. Graph.*, 7(1):76–81, 1988.

[6] Glenn L. Cash and Mehdi Hatamian. Optical character recognition by the method of moments. *Comput. Vision Graph. Image Process.*, 39(3):291–310, 1987.

[7] Cassidy J. Curtis, Sean E. Anderson, Joshua E. Seims, Kurt W. Fleischer, and David H. Salesin. Computer-generated watercolor. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 421–430, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[8] Debra Dooley and Michael F. Cohen. Automatic illustration of 3d geometric models: surfaces. In *VIS '90: Proceedings of the 1st conference on Visualization '90*, pages 307–314, Los Alamitos, CA, USA, 1990. IEEE Computer Society Press.

[9] S. Dudani, K. Breeding, and R. McGhee. Aircraft identification by moment invariants. In *IEEE Trans. Comput*, pages vol. C–26, 39–46, 1977.

[10] William T. Freeman, David B. Anderson, Paul A. Beardsley, Chris N. Dodge, Michal Roth, Craig D. Weissman, William S. Yerazunis, Hiroshi Kage, Kazuo Kyuma, Yasunari Miyake, and Ken ichi Tanaka. Computer vision for interactive computer graphics. *IEEE Comput. Graph. Appl.*, 18(3):42–53, 1998.

[11] Amy Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen. A non-photorealistic lighting model for automatic technical illustration. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 447–452, New York, NY, USA, 1998. ACM Press.

[12] Bruce Gooch, Peter-Pike J. Sloan, Amy Gooch, Peter Shirley, and

Richard F. Riesenfeld. Interactive technical illustration. In *Symposium on Interactive 3D Graphics*, pages 31–38, 1999.

[13] Stuart Green, David Salesin, Simon Schofield, Aaron Hertzmann, Peter Litwinowicz, and Amy Gooch. Non-photorealistic rendering. In *SIGGRAPH Non-Photorealistic Rendering Course Notes*, 1999.

[14] Donald P. Greenberg. A framework for realistic image synthesis. *Commun. ACM*, 42(8):44–53, 1999.

[15] Paul E. Haeberli. Paint by numbers: Abstract image representation. In *ACM SIGGRAPH 90*, volume 24, pages 207–214, 1990.

[16] A. Hartner, M. Harter, E. Cohen, and B. Gooch. Object space silhouette algorithms. In *Theory and Practice of Non-Photorealistic Graphics: Algorithms, Methods, and Production System SIGGRAPH 2003 Course Notes*, 2003.

[17] Aaron Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 453–460, New York, NY, USA, 1998. ACM Press.

[18] Aaron Hertzmann. Algorithms for rendering in artistic styles. PhD thesis, New York University, 2001.

[19] Aaron Hertzmann and Ken Perlin. Painterly rendering for video and interaction. In *NPAR '00: Proceedings of the 1st international sympo-*

sium on Non-photorealistic animation and rendering, pages 7–12, New York, NY, USA, 2000. ACM Press.

[20] Aaron Hertzmann and Denis Zorin. Illustrating smooth surfaces. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 517–526, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[21] Francis S Hill. Computer graphics : using opengl. 2nd eddtion. In *Prentice Hall*, 2001.

[22] MK Hu. Visual pattern recognition by moment invariants. In *IRE Transactions on Information Theory*, pages 8:179–187, 1962.

[23] James T. Kajiya. The rendering equation. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150, New York, NY, USA, 1986. ACM Press.

[24] A. Lake, C. Marchall, M. Harris, and M. Blackstein. Stylized rendering techniques for scalable real-time 3d animation. In *NPAR 2000*, pages 13–20, 2000.

[25] Nan Li and Zhiong Huang. A feature-based pencil drawing method. In *GRAPHITE '03: Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 135–ff, New York, NY, USA, 2003. ACM Press.

[26] Peter Litwinowicz. Processing images and video for an impressionist effect. In *SIGGRAPH '97: Proceedings of the 24th annual conference*

on Computer graphics and interactive techniques, pages 407–414, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[27] Eric B. Lum and Kwan-Liu Ma. Non-photorealistic rendering using watercolor inspired textures and illumination. In *PG '01: Proceedings of the 9th Pacific Conference on Computer Graphics and Applications*, page 322, Washington, DC, USA, 2001. IEEE Computer Society.

[28] Barbara J. Meier. Painterly rendering for animation. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 477–484, New York, NY, USA, 1996. ACM Press.

[29] R. Mukundan and K.R. Ramakrishnan. Moment functions in image analysis - theory and applications. Singapore, 1998. World Scientific Publishing Co. Pte Ltd.

[30] Diego Nehab and Luiz Velho. Multiscale moment-based painterly rendering. In *SIBGRAPI '02: Proceedings of the 15th Brazilian Symposium on Computer Graphics and Image Processing*, pages 244–251, Washington, DC, USA, 2002. IEEE Computer Society.

[31] M. Obaid, R. Mukundan, and T. Bell. Enhancement of moment based painterly rendering using connected components. In *CGIV 2006: Computer Graphics, Imaging and Visualization, IEEE Computer Society Proceeding*, Sydney, Australia, 2006.

[32] Victor Ostromoukhov. Digital facial engraving. In *SIGGRAPH '99:*

Proceedings of the 26th annual conference on Computer graphics and interactive techniques, pages 417–424, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[33] Adobe Photoshop and Image Ready CS. In *Version 8.0*, 2003.

[34] S. S. Reddi. Radial and angular moment invariants for image identification. In *IEEE trans. on Pattern Analysis and Machine Intelligence*, volume 3, pages 240–242, 1981.

[35] Michael P. Salisbury, Michael T. Wong, John F. Hughes, and David H. Salesin. Orientable textures for image-based pen-and-ink illustration. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 401–406, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[36] Michio Shiraishi and Yasushi Yamaguchi. An algorithm for automatic painterly rendering based on local source image approximation. In *NPAR '00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, pages 53–58, New York, NY, USA, 2000. ACM Press.

[37] F. W. Smith and M. H. Wright. Automatic ship photo interpretation by the method of moments. In *IEEE Trans. Comput*, pages vol. C–26, 1089–1094, 1977.

[38] Mario Costa Sousa and John W. Buchanan. Observational models of

graphite pencil materials. In *Computer Forum*, volume 19, pages 27–49, 2000.

[39] S. Schlechtweg T. Strothotte. Non-photorealistic computer graphics : modeling, rendering, and animation. San Francisco, CA : Morgan Kaufmann, 2002.

[40] L. Tateosian and C. Healey. NPR: Art enhancing computer graphics. North Carolina, USA, 2004.

[41] M. R. Teague. Image analysis via the general theory of moments. In *J. Opt. Soc. Amer*, volume 70, pages 920–930, 1980.

[42] Daniel Teece. Three dimensional interactive non-photorealistic rendering. England, 1998. PHD Thesis, University of Sheffield.

[43] Jack Tumblin and Holly Rushmeier. Tone reproduction for realistic images. *IEEE Comput. Graph. Appl.*, 13(6):42–48, 1993.

[44] Scott E. Umbaugh. Computer imaging: Digital image analysis and processing. In *SE Umbaugh, CRC Press*, 2005.

[45] Luiz Velho and Jonas de Miranda Gomes. Digital halftoning with space filling curves. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 81–90, New York, NY, USA, 1991. ACM Press.

[46] Brett Wilson and Kwan-Liu Ma. Rendering complexity in computer-generated pen-and-ink illustrations. In *NPAR '04: Proceedings of the*

3rd international symposium on Non-photorealistic animation and rendering, pages 129–137, New York, NY, USA, 2004. ACM Press.

[47] Georges Winkenbach and David H. Salesin. Rendering parametric surfaces in pen and ink. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 469–476, New York, NY, USA, 1996. ACM Press.

[48] W. H. Wong and W. C. Siu. Improved digital filter structure for fast moments computation. *IEE Proceedings - Vision, Image, and Signal Processing*, 146(2):73–79, 1995.

[49] Wai-Hong Wong, Wan-Chi Siu, and Kin-Man Lam. Generation of moment invariants and their uses for character recognition. *Pattern Recogn. Lett.*, 16(2):115–123, 1995.